ซอฟต์แวร์
อธิการ มูลสาร

15

ดอสยูสเซอร์ : การแสดงผลและการเขียนโปรแกรมด้วยดีบัก

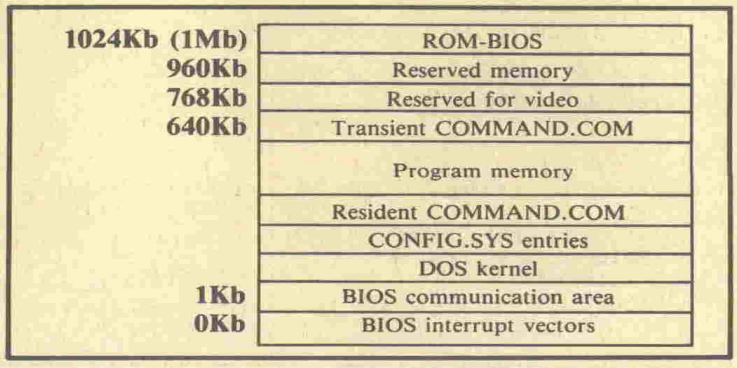
แนะนำเทคนิคน่ารู้ เกี่ยวกับการเขียนโปรแกรมขนาดเล็ก เพื่อให้ทำงานอย่างมีประสิทธิภาพสูงสุด

การแสดงผลบนเครื่องพีซีนั้น มีโหมดของการแสดงผลให้เราเลือกใช้อยู่หลายแบบ (ตารางที่ 1) เราสามารถควบคุมการแสดงผลดังกล่าวได้หลายวิธี วิธีหนึ่งที่ใช้ได้ค่อนข้างสะดวกคือ ควบคุมผ่านไดรเวอร์ ANSI.SYS ของดอส อย่างไรก็ตาม เรายังสามารถเขียนโปรแกรมเรียกใช้บริการของไบออสเพื่อการนี้ได้อีกวิธีหนึ่งด้วย ในบทความนี้จะกล่าวถึงการทำงานของทั้งสองแบบ

เราจะลองเขียนโปรแกรมที่สามารถเปลี่ยนโหมดการแสดงผลได้ตามที่เราต้องการ โปรแกรมดังกล่าวจะเป็นโปรแกรมภาษาแอสเซมบลีขนาดเล็ก ๆ ที่สร้างด้วยดีบัก สาเหตุที่สร้างด้วยดีบักได้เป็นเพราะว่าดีบักมีแอสเซมเบลอร์อย่างง่ายอยู่ในตัว

รู้จักกับ Interrupt vectors

Interrupt vectors เป็นที่เก็บชุดคำสั่งที่ต้องการใช้งาน จากรูปที่ 1 จะเห็นได้ว่าตำแหน่งของ BIOS interrupt vectors อยู่ตรง 1K แรกของหน่วยความจำในส่วนของ Convention memory และยังมี Interrupt service routine อื่น ๆ อยู่ภายในหน่วยความจำส่วนนี้รวมแล้ว 256 รูทีน ซึ่งจะมีหมายเลขกำกับไว้เริ่มตั้งแต่ 00H (H หมายถึงเลขฐานสิบหก) ไปจนถึง OFFH (บางส่วนสงวนไว้สำหรับใช้งานในอนาคต)



รูปที่ 1 โครงสร้างของหน่วยความจำ

รีจิสเตอร์ (Register)

หน่วยความจำรีจิสเตอร์คือ หน่วยความจำชนิดพิเศษที่แยกต่างหากไม่เกี่ยวข้องกับหน่วยความจำใด ๆ ในเครื่องเลย มีขนาดเล็กเพียง 1 ไบต์หรือ 1 เวิร์ด ในรูปที่ 2 จะแสดงรีจิสเตอร์พื้นฐาน 4 ตัวอันได้แก่ AX, BX, CX และ DX รีจิสเตอร์ที่กล่าวมาแต่ละตัวยังสามารถแยกอ้างถึงไบต์บน (AH, BH, CH, DH) และไบต์ล่าง (AL, BL, CL DL) ได้อีกด้วย

AX	AH	AL	**Accumulator**
BX	BH	BL	**Base**
CX	CH	CL	**Count**
DX	DH	DL	**Data**

รูปที่ 2
รีจิสเตอร์พื้นฐาน 4 ตัว

การเรียกใช้งานอินเตอร์รัพต์

การเรียกใช้ Interrupt vectors ทำได้โดยการใช้คำสั่ง INT ในภาษาแอสเซมบลี ตามด้วยหมายเลขอินเตอร์รัพต์รูทีนซึ่งอยู่ในรูปเลขฐานสิบหก อินเตอร์รัพต์แต่ละหมายเลขยังแบ่งออกเป็นฟังก์ชันได้อีกมากมาย โปรแกรมตัวอย่างที่เราจะเขียนต่อไปนี้จะเรียกใช้อินเตอร์รัพต์หมายเลข 10H (BIOS video services) และ 21H (DOS system services) เรามาดูตัวอย่างจากโปรแกรมจริงจากลิสติงที่ 1 หมายเลขบรรทัดมีไว้อ้างอิงเท่านั้น เครื่องหมาย ":" และข้อความ "<Enter>" ไม่ต้องคีย์ลงไป เสร็จแล้วเซฟเป็นเท็กซ์ไฟล์แล้วให้ชื่อว่า MODE2.SCR (Script file)

```
 1 : N MODE2.COM                <Enter>
 2 : A 100                      <Enter>
 3 : MOV AH,0                   <Enter>
 4 : MOV AL,2   ;Mode in Hex    <Enter>
 5 : INT 10                     <Enter>
 6 : MOV AH,4C                  <Enter>
 7 : INT 21                     <Enter>
 8 :                            <Enter>
 9 : R CX                       <Enter>
10 : A                          <Enter>
11 : W                          <Enter>
12 : Q                          <Enter>
```

ลิสติงที่ 1
MODE2.SCR (เปลี่ยนโหมดการแสดงผลเป็นโหมด 2)

Video Mode	Description
0	25 × 40 Monochrome Text
1	25 × 40 Color Text
2	25 × 80 Monochrome Text
3	25 × 80 Color Text
4	320 × 200 4-color Graphics
5	320 × 200 Monochrome Graphics
6	640 × 200 Monochrome Graphics
7	25 × 80 Monochrome Text
13	320 × 200 Color Graphics
14	640 × 200 16-color Graphics
15	640 × 350 Monochrome Graphics
16	640 × 350 16-color Graphics
17	640 × 480 Monochrome Graphics
18	640 × 480 VGA 16-color Graphics
19	320 × 200 VGA 256-color Graphics

ตารางที่ 1
โหมดของการแสดงผลบนเครื่องพีซีและพีซีคอมแพตทิเบิล

คุณสามารถสร้างไฟล์คำสั่ง MODE2.COM ได้โดยการใช้เทคนิคการเปลี่ยนทิศทางนำเข้าข้อมูล โดยใช้เครื่องหมาย "<" ของดอส โดยคีย์คำสั่งดังนี้

```
C:\>DEBUG < MODE2.SCR <Enter>
```

เมื่อเรียกใช้ที่ดอสพรอมต์ก็จะเปลี่ยนโหมดการแสดงผลเป็นโหมด 2 ตามต้องการ

คำสั่ง N (Name)

ในบรรทัดที่ 1 คำสั่ง N เป็นคำสั่งของดีบักซึ่งเป็นตัวอักษร 1 ตัวเสมอ ระบุชื่อไฟล์ที่เราจะต้องติดต่อด้วย เราต้องการสร้างไฟล์คำสั่งชื่อ MODE2.COM ที่จะเปลี่ยนโหมดการแสดงผลเป็นแบบ 25×80 Monochrome Text

คำสั่ง A (Assemble)

การเขียนโปรแกรมบนดีบักนั้นจะเขียนอยู่ในรูปภาษาแอสเซมบลี ซึ่งเป็นรหัสย่อที่จะนำไปแปลเป็นภาษา-

เครื่องโดยแอสเซมเบลอร์ในดีบัก คำสั่งนี้จะบอกให้ดีบักรู้ว่าเราต้องการใช้งานแอสเซมเบลอร์ ในบรรทัดที่ 2 จะสังเกตว่ามีเลข 100 ตามท้ายคำสั่งมาด้วย ตัวเลขนี้คือ ตำแหน่งบนหน่วยความจำ (แอดเดรส) ที่จะเริ่มทำงาน ซึ่งอยู่ในรูปเลขฐานสิบหก เพราะดีบักจะทำงานกับตัวเลขฐานสิบหกเท่านั้น โปรแกรมที่สร้างด้วยดีบักและโปรแกรมแอสเซมบลีอื่น ๆ มักจะเริ่มต้นที่แอดเดรสนี้เสมอ

คำสั่ง MOV (e)

เป็นคำสั่งภาษาแอสเซมบลี ถ้าคุณป้อนคำสั่งทั้งหมดจากลิสติงที่ 1 ลงดีบักโดยตรงคุณจะพบว่าพรอมต์ของดีบักจะเปลี่ยนไปโดยอยู่ในรูป XXXX: YYYY (XXXX) เป็นตัวเลขใด ๆ ที่ระบุตำแหน่งเซกเมนต์ ซึ่งไม่สำคัญต่อการทำงานของเรา YYYY คือตำแหน่งของออฟเซตในหน่วยความจำ มันจะเพิ่มขึ้นเรื่อย ๆ ตามขนาดของชุดคำสั่ง รูปที่ 3 จะแสดงผลลัพธ์ที่ปรากฏทางหน้าจอที่ดีบักแสดงออกมา

จากลิสติงที่ 1 บรรทัดที่ 3, 4 และ 6 มีคำสั่ง MOV อยู่ด้วย ในบรรทัดที่ 1 เป็นคำสั่งให้นำค่า 0 ไปเก็บไว้ในรีจิสเตอร์ AH โปรแกรมภาษาแอสแซมบลีที่ต้องการเรียกใช้ BIOS interrupt vectors จะต้องระบุหมายเลขฟังก์ชันการทำงานไว้ในรีจิสเตอร์นี้ (ศึกษาได้จากคู่มือ) ซึ่งก็คือฟังก์ชันหมายเลข 0 บรรทัดที่ 2 นำค่า 2 ไปไว้ในรีจิสเตอร์ AL รีจิสเตอร์นี้จะเก็บโหมดของการแสดงผลที่เราต้องการ ซึ่งก็คือโหมด 2

```
-N MODE2.COM
-A 100
634F:0100 MOV AH,0
634F:0102 MOV AL,2
634F:0104 INT 10
634F:0106 MOV AH,4C
634F:0108 INT 21
634F:010A
-R CX
CX 0000
:A
-W
Writing 0000A bytes
-Q
```

รูปที่ 3 ผลลัพธ์ที่ดีบักแสดงออกมาทางหน้าจอ

คำสั่ง INT (errupt)

การอินเตอร์รัพต์คือ การขัดจังหวะการทำงาน แต่ในภาษาแอสเซมบลีแล้วจะหมายถึงการเรียกใช้โปรแกรมย่อย (Interrupt service routine) บนเครื่องไมโครคอมพิวเตอร์มีหมายเลขอินเตอร์รัพต์ให้เราเลือกใช้อยู่มากมาย อินเตอร์รัพต์ที่ใช้บ่อยมาก ๆ คือ อินเตอร์รัพต์หมายเลข 10H และหมายเลข 21H ตารางที่ 2 จะแสดงหมายเลขของอิน-

ตารางที่ 2 หมายเลขอินเตอร์รัพต์และวัตถุประสงค์ของการใช้งาน

No.	วัตถุประสงค์	No.	วัตถุประสงค์
00H	Hardware divide by zero	1DH	Video parameter table
01H	Hardware single step trap	1EH	Disk parameter table
		1FH	Graphics character table
02H	Nonmaskable interrupt	20H	DOS terminate program
03H	Debugger breakpoint set	21H	DOS system services
04H	Arithmetic overflow	22H	Program terminate
05H	BIOS print screen	23H	DOS CTRL-BREAK
08H	IRQ0 clock tick	24H	DOS critical-error handler
09H	IRQ1 keyboard		
0AH	IRQ2	25H	DOS disk read
0BH	IRQ3 COM2	26H	DOS disk write
0CH	IRQ4 COM1	27H	DOS terminate resident
0DH	IRQ5 PC/XT harddisk or IRQ5 PC/AT LPT2	28H	DOS idle
		29H	DOS fast putchar
0EH	IRQ6 diskette	2AH	MS-Net services
0FH	IRQ7 LPT1	2EH	DOS primary shell program loader
10H	BIOS video services		
11H	BIOS equipment list service	2FH	DOS multiplex interrupt
		33H	Mouse services
12H	BIOS memory size service	40H	Diskette vector if harddisk present
13H	BIOS disk services	41H	Harddisk parameter table
14H	BIOS communication services	42H	EGA BIOS redirection
		43H	EGA parameter table
15H	BIOS miscellaneous services	44H	EGA character table
		4AH	PC/AT Int 70H alarm
16H	BIOS keyboard service	5CH	NetBIOS services
17H	BIOS printer services	67H	EMS services
18H	Invoke ROM-BASIC	70H	IRQ8 PC/AT real-time clock
19H	System reboot		
1AH	BIOS time-of-day services	71H	IRQ9 PC/AT redirect of Int 0AH
1BH	CTRL-BREAK handler	75H	IRQ13 PC/AT math coprocessor
1CH	Called by int 08H handler		

เตอร์รัพต์และวัตถุประสงค์การใช้งาน

บรรทัดที่ 5 ของลิสติงที่ 1 เมื่อเราใส่หมายเลขฟังก์ชันและโหมดการแสดงผลในรีจิสเตอร์ AH และ AL ตามลำดับแล้ว เราก็พร้อมเรียกใช้บริการของอินเตอร์รัพต์ 10H ฟังก์ชัน 0 ได้ทันที

บรรทัดที่ 6 และ 7 เราจะใส่หมายเลขฟังก์ชัน 4CH ในรีจิสเตอร์ AH แล้วเรียกใช้อินเตอร์รัพต์หมายเลข 21H เพื่อบอกดอสให้รู้ว่าเราจบการทำงานที่บรรทัดนี้ (เหมือนกับคำสั่ง end. ในภาษาปาสคาลหรือคำสั่ง end ในภาษาเบสิก) แล้วกลับสู่ระบบ

บรรทัดที่ 8 เรากดปุ่ม Enter ผ่านเพื่อบอกให้ดีบักทราบว่าเราไม่มีข้อมูลที่จะคีย์อีก เมื่อดีบักพบบรรทัดว่างก็จะรู้ได้ทันทีว่าจบการป้อนคำสั่งภาษาแอสเซมบลีแล้ว ดีบักจะแสดงเครื่องหมายพร้อมรับคำสั่งของตนเอง (เครื่องหมาย "–") และทำงานต่อไป

คำสั่ง R (Register)

เป็นคำสั่งของดีบัก ระบุรีจิสเตอร์ที่เราต้องการนำข้อมูลไปเก็บ บรรทัดที่ 9 และ 10 จะนำค่า 0AH ไปเก็บในรีจิสเตอร์ CX ซึ่งรีจิสเตอร์นี้จะเก็บขนาดของโปรแกรมที่เราเพิ่งป้อนผ่านไป ถ้าเราไม่ระบุค่าในรีจิสเตอร์ CX เมื่อเราทำการเขียนคำสั่งที่แปลผ่านแอสเซมเบลอร์ของดีบักลงบนแผ่นดิสค์ ก็จะได้ขนาดที่ไม่แน่นอน ขึ้นอยู่กับค่าที่เก็บอยู่ใน CX และอาจมีผลเสียต่อการทำงานได้เป็นอย่างมาก จากที่ผ่านมาโปรแกรมของเรามีขนาดเพียง 10 ไบต์ เท่านั้น (0AH ไบต์)

คำสั่ง W (Write) และ Q (Quit)

จะเขียนชุดคำสั่งทั้งหมดลงบนดิสค์แล้วออกจากการทำงานของดีบักกลับสู่ดอส จะเห็นได้ว่าค่อนข้างยุ่งยากพอสมควรในการเขียนโปรแกรมภาษาแอสเซมบลี อย่างไรก็ตาม รหัสที่ได้จะมีขนาดกระทัดรัดเพียง 10 ไบต์ และทำงานได้เร็วที่สุดเมื่อเทียบกับภาษาอื่น ยกเว้นภาษาเครื่อง เพราะทุกคำสั่งจะทำงานกับระบบโดยตรง ไม่ต้องผ่านตัวจัดการเหมือนในภาษาโปรแกรมทั่วไป ซึ่งถึงแม้จะแปลเป็นภาษาเครื่องแล้วก็ยังได้รหัสคำสั่งไม่กระทัดรัดภาษาแอสเซมบลีจึงเหมาะกับงานที่มีขนาดเล็ก ๆ และต้องการความเร็วสูง

ได้หนึ่งโปรแกรมล่ะ...อยากได้มากกว่านี้

ตอนนี้เราได้โปรแกรม MODE2.COM ไว้ใช้งานแล้ว ลองเปลี่ยนจากหมายเลข 2 ในบรรทัดที่ 4 เป็นเลขอื่น ๆ โดยดูหมายเลขได้จากตารางที่ 1 อย่าลืมว่าต้องเป็นเลขฐานสิบหกเท่านั้น เปลี่ยนชื่อไฟล์ในบรรทัดแรกเป็น MODEn.COM ตามหมายเลขโหมดด้วย คุณก็จะได้โปรแกรมหลาย ๆ โปรแกรมมาใช้ เปลี่ยนโหมดกลับไปกลับมาทดลองดูแล้วคุณจะเห็นได้ว่า เราสามารถเปลี่ยนเป็นโหมดกราฟิกได้ด้วย ถึงตอนนี้นักเล่นเกมที่เคยโรมรันกับโปรแกรม HCG ที่เดี๋ยวดีเดี๋ยวร้าย คงจะนึกออกว่าไม่ต้องพึ่งมันอีกต่อไปแล้ว นอกจากนี้ในโหมดกราฟิกผมลอง Type ไฟล์ที่เก็บอักขระกราฟิกที่เป็นกรอบ ตาราง และสัญลักษณ์พิเศษต่าง ๆ ออกมา พบว่าได้ตัวอักษรที่สวยงามไปอีกแบบหนึ่ง น่าลองดูนะครับ

เปลี่ยนโหมดการแสดงผลด้วย ANSI.SYS

หากได้ติดตั้ง ANSI.SYS เป็นที่เรียบร้อยแล้ว เราสามารถเปลี่ยนโหมดการแสดงผลได้อย่างง่ายดาย (ตารางที่ 3) การป้อนรหัส {Esc} ทำได้โดยป้อนผ่านโปรแกรมเอดิเตอร์ที่ยอมรับรหัส แต่ในบทความนี้เพื่อเป็นการทดสอบ

Mode	Description	Escape Sequence
0	25 × 40 Monochrome Text	{Esc}[=0h
1	25 × 40 Color Text	{Esc}[=1h
2	25 × 80 Monochrom Text	{Esc}[=2h
3	25 × 80 Color Text	{Esc}[=3h
4	320 × 200 4-color Graphics	{Esc}[=4h
5	320 × 200 Monochrome Graphics	{Esc}[=5h
6	640 × 200 Monochrome Graphics	{Esc}[=6h
7	Line wrap enable	{Esc}[=7h
7	Line wrap disable	{Esc}[=7l
13	320 × 200 Color Graphics	{Esc}[=13h
14	640 × 200 16-color Graphics	{Esc}[=14h
15	640 × 350 Monochrome Graphics	{Esc}[=15h
16	640 × 350 16-color Graphics	{Esc}[=16h
17	640 × 480 Monochrome Graphics	{Esc}[=17h
18	640 × 480 VGA 16-color Graphics	{Esc}[=18h
19	320 × 200 VGA 256-color Graphics	{Esc}[=19h

ตารางที่ 3
คำสั่งเปลี่ยนโหมดการแสดงผลของ ANSI.SYS

การเขียนโปรแกรมด้วยดีบัก เราจะเขียนโปรแกรมอีกโปรแกรมหนึ่งที่สามารถส่งรหัส {Esc} ได้โดยตรง โดยที่ไม่ต้องป้อนผ่านเอดิเตอร์อีกต่อไป

ESCAPE.COM

ลิสติงที่ 2 จะแสดง ESCAPE.SCR ที่ใช้สร้าง ESCAPE.COM ในโปรแกรมนี้เราจะได้รู้จักคำสั่งต่าง ๆ เพิ่มขึ้นมาอีก และคุณยังจะได้รู้จักวิธีการจัดการกับบรรทัดคำสั่ง (Command line) อีกด้วย

เช่นเดียวกับโปรแกรมที่แล้ว ให้สร้าง ESCAPE.COM ผ่านดีบักแล้วทดสอบดู ผมจะสมมติตัวอย่างให้ ถ้าคุณติดตั้ง ANSI.SYS แล้วคุณอยากจะเปลี่ยนแอตทริบิวต์การแสดงผลเป็นแบบรีเวอร์สวิดิโอ ({Esc}[7m) ลองพิมพ์คำสั่งดังนี้

```
C:\ >ESCAPE [7m <Enter>
```

สะดวกกว่าเดิมเยอะจริงไหมครับ อ้อ! อย่าลืมเปลี่ยนกลับล่ะเดี๋ยวจะลายตาเสียก่อน หลังจากนี้คุณนำไปใส่แบตช์ไฟล์ได้ตามปกติ

การจัดการกับบรรทัดคำสั่ง

เคยสังเกตไหมว่าคำสั่งที่คุณคีย์ลงไปที่ดอสพรอมต์นั้น ดอสจัดการกับมันอย่างไรบ้าง ทุก ๆ ครั้งที่คุณป้อนคำสั่งเสร็จ ดอสจะสร้างบัฟเฟอร์ขึ้นมา มันมีขนาดเล็กเพียง 256 ไบต์ (0FFH) เท่านั้น เราเรียกบัฟเฟอร์นี้ว่า Program Segment Prefix (PSP) ซึ่งจะถูกโหลดไปอยู่ที่ตำแหน่งก่อนหน้าโปรแกรมที่คุณรันนั่นแหละ (ดูรูปที่ 4 ประกอบ) คงหายสงสัยแล้วว่าทำไมโปรแกรมที่สร้างจึงต้องเริ่มต้นที่แอดเดรส 100H

เนื้อที่ส่วน PSP จะถูกใช้ในการเก็บข่าวสารอะไรต่อมิอะไรมากมาย บรรทัดคำสั่งที่คุณป้อนลงไปก็อยู่ใน PSP นี่แหละ โดยเราจะดูกันที่แอดเดรส 80H ผมสมมติว่าผมคีย์คำสั่ง "ESCAPE Test Cmd__Line<Enter>" ผมลองใช้ดีบักติดตามดูข้อมูลที่แอดเดรสดังกล่าว พบว่ามีข้อมูลดังรูปที่ 5

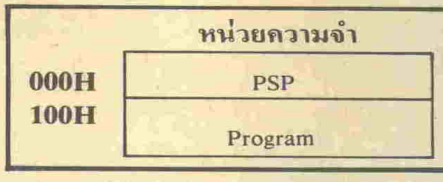
จะสังเกตเห็นได้ชัดเจนว่าข้อความที่คุณป้อนตามท้ายคำสั่งเข้าไป ถูกเก็บไว้ตามลำดับเริ่มต้นที่แอดเดรส 82H และยังมีความยาว 0EH ไบต์ เก็บไว้ที่แอดเดรส 80H ด้วยและปิดท้ายข้อมูลด้วยอักขระแอสกีตัวที่ 13 (Carriage Return) หรือ <Enter> นั่นเอง ส่วนที่ว่างที่แอดเดรส 81H นั้นคือ ช่องว่างที่แบ่งระหว่างคำสั่ง (ESCAPE) กับพารามิเตอร์ (Test Cmd__Line)

มาไล่โปรแกรมกันสักนิด

จากลิสติงที่ 2 บรรทัดที่ 1 และ 2 จะขอข้ามไป บรรทัดที่ 3-5 เรียกใช้อินเตอร์รัพต์ 21H ต่อไปนี้จะเรียกว่าดอสอินเตอร์รัพต์ ฟังก์ชัน 2H เพื่อ อินเตอร์รัพต์ 21H (ต่อไปนี้จะเรียกว่าดอสอินเตอร์รัพต์) ฟังก์ชัน 2H เพื่อพิมพ์อักขระ 1 ตัวคือ อักขระ {Esc} หรือ 1BH จะสังเกตว่าต่างจากโปรแกรมแรก (MODE2.COM) ตรงที่ตัวอักษรที่จะพิมพ์กลับนำไปใส่ในรีจิสเตอร์ DL แทนที่จะเป็น AL ทั้งที่เป็นอินเตอร์รัพต์การพิมพ์เหมือนกัน ทั้งนี้เพราะ

ลิสติงที่ 2
ESCAPE.SCR
(แสดงผลรหัส Escape)

```
1 : N ESCAPE.COM          <Enter>
2 : A 100                 <Enter>
3 : MOV AH,2              <Enter>
4 : MOV DL,1B             <Enter>
5 : INT 21                <Enter>
6 : MOV BX,80             <Enter>
7 : INC BX                <Enter>
8 : CMP BYTE [BX],D       <Enter>
9 : JNE 109               <Enter>
10 : MOV BYTE [BX],24     <Enter>
11 : MOV AH,9             <Enter>
12 : MOV DX,82            <Enter>
13 : CMP DX,BX            <Enter>
14 : JG 11D               <Enter>
15 : INT 21               <Enter>
16 : MOV AH,4C            <Enter>
17 : INT 21               <Enter>
18 :                      <Enter>
19 : R CX                 <Enter>
20 : 21                   <Enter>
21 : W                    <Enter>
22 : Q                    <Enter>
```



รูปที่ 4 ตำแหน่งของ PSP ในหน่วยความจำ

OEH		T	e	s	t		C	m	d	_	L	i	n	e	ODH
80H	81H	82H													

รูปที่ 5 ข้อมูลของตำแหน่ง 80H หลังจากป้อนคำสั่ง "C:\>ESCAPE Test Cmd_Line<Enter>"

เป็นข้อกำหนดของแต่ละอินเตอร์รัพต์ สำหรับดอสอินเตอร์รัพต์นั้นจะใช้รีจิสเตอร์ DX เป็นที่เก็บข้อมูล ส่วนไบออสอินเตอร์รัพต์จะใช้รีจิสเตอร์ AL แทน

รีจิสเตอร์ที่นับว่ามีความสำคัญมากคือ รีจิสเตอร์ BX เพราะเป็นรีจิสเตอร์พื้นฐานที่ชี้ไปยังตำแหน่งของข้อมูลในหน่วยความจำบรรทัดที่ 6 จึงให้ BX ชี้ไปที่แอดเดรส 80H ซึ่งเรากำลังสนใจ บรรทัดที่ 7-9 จะวนลูปการทำงานจนกว่าค่าที่เก็บอยู่ในแอดเดรสที่ BX ชี้อยู่จะเท่ากับ 0DH (Carriage Return) ก็จะถือว่าจบการวนลูป คำสั่ง INC จะเพิ่มค่าให้กับรีจิสเตอร์ คำสั่ง CMP จะเปรียบเทียบระหว่างค่าสองค่า (คงพอเดาได้) และคำสั่ง JNE (Jump if Not Equal) จะกระโดดไปทำงานที่แอดเดรสที่กำหนดถ้าผลการเปรียบเทียบออกมาได้ไม่เท่ากัน ลองดู Pseudo code ของ 3 บรรทัดนี้กัน

บรรทัดที่ 7 : เพิ่มค่าให้กับรีจิสเตอร์ BX (ดีฟอลต์คือ เพิ่มค่า 1 สามารถระบุได้)

บรรทัดที่ 8 : เปรียบเทียบค่าที่เก็บในแอดเดรสที่ BX ชี้อยู่ว่าเท่ากับ 0DH หรือไม่

บรรทัดที่ 9 : ถ้าไม่เท่ากัน กลับไปทำที่บรรทัดที่ 7

การทำงานในบรรทัดที่ 7-9 ออกจะทำให้เรารู้สึกงงอยู่ไม่น้อย แต่ถ้าลองติดตามดูต่อไปจะเข้าใจ บรรทัดที่ 10 นำค่า 24H (มีค่าแอสกีเท่ากับตัวอักษร $) ไปใส่ในแอดเดรสที่ BX ชี้อยู่ ซึ่งก็คือการนำเครื่องหมาย $ ไปแทนรหัส Carriage Return หรือ <Enter> เพื่อที่ว่าอินเตอร์รัพต์ 21H ฟังก์ชัน 9 ซึ่งมีหน้าที่พิมพ์สายวลี ASCIIZ ซึ่งจะต้องจบลงด้วยอักษร $ จะทำงานได้ถูกต้อง

บรรทัดที่ 11 ใส่หมายเลขฟังก์ชันคือ 9 ใน AH บรรทัดที่ 12 ใส่ค่า 82H ใน DX บรรทัดที่ 13 จะเป็นการเปรียบเทียบตำแหน่งจุดเริ่มต้น (DX) กับตำแหน่งสุดท้าย (BX) ของข้อมูล ถ้าจุดเริ่มต้น (82H) มากกว่าจุดสุดท้าย (ในกรณีไม่มีข้อมูลตามท้ายคำสั่งค่าใน BX จะได้ 81H ลองไล่ดูจะรู้) เพื่อป้องกันการเกิดปัญหา บรรทัดที่ 14 JG (Jump if Greater) จึงกระโดดไปที่บรรทัดที่ 16 แทน (บรรทัดที่ 14) แต่ถ้าไม่เป็นเช่นนั้นก็พิมพ์สายวลีทั้งหมดออกมา บรรทัดที่ 15 เรียกใช้อินเตอร์รัพต์

บรรทัดที่ 16-17 สั่งจบการทำงาน

ไล่โปรแกรมกันเสียเหนื่อยเลยเป็นอย่างไรบ้างครับ ความรู้เล็กๆ น้อยๆ เหล่านี้ผมก็ค้นคว้าด้วยตัวผมเองนี่แหละ ผิดพลาดประการใดก็ขออภัยมา ณ ที่นี้ด้วย ผมมักจะนำความรู้เรื่องการเขียนโปรแกรมเล็ก ๆ ด้วยดีบักเหล่านี้ไปประยุกต์ใช้กับการโปรแกรมแบตช์ไฟล์ของผม เพราะมันจะช่วยเสริมกันได้ดี สิ่งไหนแบตช์ไฟล์ขาด ดีบักทำแทนได้ อย่างนี้ผมจะทิ้งแบตช์ไฟล์ไปได้หรือ

อย่าลืมกลับไปลองเปิดคู่มือดอสเอาแบบแอดวานซ์หน่อยนะครับ ไล่ดูฟังก์ชันการทำงานของแต่ละอินเตอร์รัพต์แล้วคุณจะเกิดความคิดอยากทำโปรแกรมโน้น อยากทำโปรแกรมนี้ เยอะแยะไปหมด ขอให้สนุกครับ

ก่อนจาก

เทคนิคการโปรแกรมอินเตอร์รัพต์ยังมีอีกมากมายนัก ที่ยกตัวอย่างมาเป็นเพียงส่วนน้อยนิด อย่างไรก็ตาม ถ้าหากสนใจการเขียนโปรแกรมภาษาแอสเซมบลีก็ลองศึกษาดู คุณจะพบว่าโลกของพีซียังมีอะไรที่คุณยังไม่รู้อีกเยอะ และมันก็น่าตื่นเต้นมากด้วย คุณรู้ไหมว่าไมโครคอมพิวเตอร์ที่คุณใช้อยู่มีวิธีจัดการกับอุปกรณ์ของมันอย่างไร จอภาพ คีย์บอร์ด เครื่องพิมพ์ เม้าส์ ดิสก์ไดรฟ์ ฯลฯ คุณจะเขียนโปรแกรมอย่างไรให้มีประสิทธิภาพสูงสุด ทำงานได้เร็วที่สุด เรื่องอย่างนี้มันช่างท้าทายเสียจริง ๆ...